Fine-grained Mobility (in Emerald)

Robert Grimm
New York University
Altogether Now: The Three Questions

- What is the problem?
- What is new or different?
- What are the contributions and limitations?
Why Migration?

- Migration in general
  - Load sharing
  - Communications performance
  - Availability
  - Reconfiguration
  - Utilizing special capabilities

- Fine-grained object mobility
  - Data movement
  - Invocation performance
  - Garbage collection
Emerald Goals

- Provide mobility without sacrificing performance
  - Procedure calls in the local case
  - RPC in the remote case
- Provide a single object model
  - While still allowing for different implementations
- Target environment: Local network with up to 100 nodes
Emerald Objects

- Objects have four components
  - Network-wide name
  - Representation: data & references to other objects
  - Set of operations
  - Optional process

- Objects are *not* class-based, do *not* form a hierarchy
  - They have a concrete type object (which has code)
  - They can be compared against an abstract type (interface)
  - What are the advantages/disadvantages of this instance-based object model?
Emerald Mobility

- Five primitives
  - Locate, move, fix, unfix, refix
- Explicit location (node object)
- Implicit location (any other object)
- Attachments
  - Control what objects are moved
  - Are transitive
  - Are not symmetric
Emerald Calling Conventions

- In general: Call-by-reference
- Automatic argument moving
  - Controlled by compiler (think, small immutable objects)
  - Controlled by programmer
    - Call-by-move
    - Call-by-visit
Emerald Processes

- Stacks of activation records
- Object may move → What to do about activations?
  - Always return to original node
    - Leaves residual dependencies
  - Move activations with objects
    - Need to be clever in implementation
Implementation
Three Types of Addressing Structures

- Global, local, and direct objects
  - What happens if a global object is on a different node?
Finding Objects

- Mechanism based on *forwarding addresses*
  - Each object has a global object identifier (OID)
  - Each node has a table mapping OIDs to object descriptors
    - `<timestamp, node>, OID`
  - Every sent reference contains OID and forwarding address
  - Searching node follows up to two forwarding addresses
    - If that does not yield the location, it broadcasts a search msg
- Why not keep a directory of nodes referencing an object?
Finding and Translating Pointers

- **Problem:** Emerald uses direct addresses
  - Local to a machine, need to be translated on move
- **Solution:** Object and activation record templates
  - Identify locations and types of pointers

```c
const simpleobject == object simpleobject
  monitor
    var myself : Any ← simpleobject
    var name : String ← “Emerald”
    var i : Integer ← 17
    operation GetMyName → [n : String]
      n ← name
    end GetMyName
  :
end monitor
end simpleobject
```
Moving Objects

- **Moving data objects**
  - Messages include data area, translation information, and OID & forwarding address for global object pointers
  - Receiver allocates space, builds translation table, makes sure object descriptors exist, traverses data (templates!)

- **Moving activation records**
  - Problem: need to locate activation records for object
  - Possible solutions
    - Record invocations → Too expensive on regular invocations
    - Search invocations → Too expensive on moves
Moving activation records (cont.)

- Emerald solution
  - Maintain list of activation records
  - On invocation, mark activation record as “not linked”
  - On preemption, traverse stack for not linked records and link them
  - Why is this cheaper than recording invocations?

Handling processor registers

- Emerald uses callee-saved registers → Why?
- All registers need to be included in moving activation record
Garbage Collection

- Two collectors: one local and one global
- Global collector
  - Builds on object descriptors
    - Represent out edges (already)
  - Implements mark-and-sweep algorithm
    - Pains objects white, gray, and black → What do colors mean?
  - Uses some clever techniques to deal with mobility and concurrency
Performance
### Microbenchmarks

**Table II. Remote Operation Timing**

<table>
<thead>
<tr>
<th>Operation type</th>
<th>Time/ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local invocation</td>
<td>0.019</td>
</tr>
<tr>
<td>Kernel CPU time, remote invocation</td>
<td>3.4</td>
</tr>
<tr>
<td>Elapsed time, remote invocation</td>
<td>27.9</td>
</tr>
<tr>
<td>Remote invocation, local reference parameter</td>
<td>31.0</td>
</tr>
<tr>
<td>Remote invocation, call-by-move parameter</td>
<td>33.0</td>
</tr>
<tr>
<td>Remote invocation, call-by-visit parameter</td>
<td>37.4</td>
</tr>
<tr>
<td>Remote invocation, remote reference parameter</td>
<td>61.8</td>
</tr>
</tbody>
</table>

- What do we learn from this table?
Messaging

What do we learn from this experiment?

<table>
<thead>
<tr>
<th></th>
<th>Without mobility</th>
<th>With mobility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total elapsed time (in seconds)</td>
<td>71</td>
<td>55</td>
</tr>
<tr>
<td>Remote invocations</td>
<td>1,386</td>
<td>666</td>
</tr>
<tr>
<td>Network messages sent</td>
<td>2,772</td>
<td>1,312</td>
</tr>
<tr>
<td>Network packets sent</td>
<td>2,940</td>
<td>1,954</td>
</tr>
<tr>
<td>Total bytes transferred</td>
<td>568,716</td>
<td>528,696</td>
</tr>
<tr>
<td>Total bytes moved</td>
<td>0</td>
<td>382,848</td>
</tr>
</tbody>
</table>
What Do You Think?